

# Vision Paper: Towards an Understanding of the Limits of Map-Reduce Computation

Foto N. Afrati<sup>†</sup>, Anish Das Sarma<sup>#</sup>, Semih Salihoglu<sup>‡</sup>, Jeffrey D. Ullman<sup>‡</sup>

<sup>†</sup> National Technical University of Athens, <sup>#</sup> Google Research, <sup>‡</sup> Stanford University  
afrati@softlab.ece.ntua.gr, anish.dassarma@gmail.com, semih@cs.stanford.edu, ullman@gmail.com

## 1. INTRODUCTION

A significant amount of recent research work has addressed the problem of solving various data management problems in the cloud. The major algorithmic challenges in map-reduce computations involve balancing a multitude of factors such as the number of machines available for mappers/reducers, their memory requirements, and *communication cost* (total amount of data sent from mappers to reducers). Most past work provides custom solutions to specific problems, e.g., performing fuzzy joins in map-reduce [2, 8], clustering [3], graph analyses [1, 6, 5], and so on. While some problems are amenable to very efficient map-reduce algorithms, some other problems do not lend themselves to a natural distribution, and have provable lower bounds. Clearly, the ease of “map-reducibility” is closely related to whether the problem can be partitioned into independent pieces, which are distributed across mappers/reducers. What makes a problem distributable? Can we characterize general properties of problems that determine how easy or hard it is to find efficient map-reduce algorithms?

This is a vision paper that attempts to answer the questions described above. We define and study *replication rate*. Informally, the replication rate of any map-reduce algorithm gives the average number of reducers each input is sent to. There are many ways to implement nontrivial problems in a round of map-reduce; the more parallelism you want, the more overhead you face due to having to replicate inputs to many reducers. In this paper:

- We offer a simple model of how inputs and outputs are related, enabling us to study the replication rate of problems. We show how our model can capture a varied set of problems. (Section 2)
- We study two interesting problems—*Hamming Distance-1* (Section 3) and *triangle finding* (Section 4)—and show in each case there is a lower bound on the replication rate that grows as the number of inputs per reducer shrinks (and therefore as the parallelism grows). Moreover, we present methods of mapping inputs to reducers that meet these lower bounds for various values of inputs/reducer.

It is our long-term goal to understand how the structure of a problem, as reflected by the input-output relationship in our model, affects the degree of parallelism/replication tradeoff.

## 2. THE MODEL

The model looks simple – perhaps too simple. But with it we can discover some quite interesting and realistic insights into the range of possible map-reduce algorithms for a problem. For our purposes, a *problem* consists of:

1. Sets of *inputs* and *outputs*.
2. A *mapping* from outputs to sets of inputs. The intent is that each output depends on only the set of inputs it is mapped to.

Note that our model essentially captures the notion *provenance* [7]. In our context, there are two nonobvious points about this model:

- Inputs and outputs are hypothetical, in the sense that they are all the possible inputs or outputs that might be present in an instance of the problem. Any *instance* of the problem will have a subset of the inputs. We assume that an output is never made unless at least one of its inputs is present, and in many problems, we only want to make the output if *all* of its associated inputs are present.
- We need to limit ourselves to finite sets of inputs and outputs. Thus, a finite domain or domains from which inputs and outputs are constructed is often an integral part of the problem statement, and a “problem” is really a family of problems, one for each choice of finite domain(s).

We hope a few examples will make these ideas clear.

### 2.1 Examples of Problems

**EXAMPLE 2.1.** Consider the natural join of relations  $R(A, B)$  and  $S(B, C)$ . The inputs are tuples in  $R$  or  $S$ , and the outputs are tuples with schema  $(A, B, C)$ . To make this problem finite, we need to assume finite domains for attributes  $A$ ,  $B$ , and  $C$ ; say there are  $N_A$ ,  $N_B$ , and  $N_C$  members of these domains, respectively.

Then there are  $N_A N_B N_C$  outputs, each corresponding to a triple  $(a, b, c)$ . This output is mapped to the set of two inputs. One is the tuple  $R(a, b)$  from relation  $R$  and the other is the tuple  $S(b, c)$  from relation  $S$ . The number of inputs is  $N_A N_B + N_B N_C$ .

Notice that in an instance of the join problem, not all the inputs will be present. That is, the relations  $R$  and  $S$  will be subsets of all the possible tuples, and the output will be those triples  $(a, b, c)$  such that both  $R(a, b)$  and  $S(b, c)$  are actually present in the input instance.

**EXAMPLE 2.2.** For another example, consider finding triangles. We are given a graph as input and want to find all triples of nodes such that in the graph there are edges between each pair of these three nodes. To model this problem, we need to assume a domain for the nodes of the input graph with  $N$  nodes. An output is thus a set of three nodes, and an input is a set of two nodes. The output  $\{u, v, w\}$  is mapped to the set of three inputs  $\{u, v\}$ ,  $\{u, w\}$ , and  $\{v, w\}$ . Notice that, unlike the previous and next examples, here, an output is a set of more than two inputs. In an instance of the triangles problem, some of the possible edges will be present, and the

outputs produced will be those such that all three edges to which the output is mapped are present.

**EXAMPLE 2.3.** *This example is a very simple case of a similarity join. The inputs are binary strings, and since we have to make things finite, we shall assume that these strings have a fixed length  $b$ . There are thus  $2^b$  inputs. The outputs are pairs of inputs that are at Hamming distance 1; that is, the inputs differ in exactly one bit. There are thus  $(b/2)2^b$  outputs, since each of the  $2^b$  inputs is Hamming distance 1 from exactly  $b$  other inputs – those that differ in exactly one of the  $b$  bits. However, that observation counts every pair of inputs at distance 1 twice, which is why we must divide by 2.*

**EXAMPLE 2.4.** *Suppose we have a relation  $R(A, B)$  and we want to implement group-by-and-sum:*

```
SELECT A, SUM(B)
FROM R
GROUP BY A;
```

*We must assume finite domains for  $A$  and  $B$ . An output is a value of  $A$ , say  $a$ , chosen from the finite domain of  $A$ -values, together with the sum of all the  $B$ -values. This output is associated with a large set of inputs: all tuples with  $A$ -value  $a$  and any  $B$ -value from the finite domain of  $B$ . In any instance of this problem, we do not expect that all these tuples will be present, but as long as at least one of them is present, there will be an output for this value  $a$ .*

## 2.2 Mapping Schemas and Replication Rate

For many problems, there is a tradeoff between the number of reducers to which a given input must be sent and the number of inputs that can be sent to one reducer. It can be argued that the existence of such a tradeoff is tantamount to the problem being “not embarrassingly parallel”; that is, the more parallelism we introduce, the greater will be the total cost of computation.

The more reducers that receive a given input, the greater the communication cost for solving an instance of a problem using map-reduce. As communication tends to be expensive, and in fact is often the dominant cost, we’d like to keep the number of reducers per input low. However, there is also a good reason to want to keep the number of inputs per reducer low. Doing so makes it likely that we can execute the Reduce task in main memory. Also, the smaller the input to each reducer, the more parallelism there can be and the lower will be the wall-clock time for executing the map-reduce job (assuming there is an adequate number of compute-nodes to execute all the Reduce tasks in parallel).

In our discussion, we shall use the convention that  $p$  is the number of reducers used to solve a given problem instance, and  $q$  is the maximum number of inputs that can be sent to any one reducer. We should understand that  $q$  counts the number of potential inputs, regardless of which inputs are actually present for an instance of the problem. However, on the assumption that inputs are chosen independently with fixed probability, we can expect the number of actual inputs at a reducer to be  $q$  times that probability, and there is a vanishingly small chance of significant deviation for large  $q$ . If we know the probability of an input being present in the data is  $x$ , and we can tolerate  $q_1$  real inputs at a reducer, then we can use  $q = q_1/x$  to account for the fact that not all inputs will actually be present.

With this motivation in mind, let us define a *mapping schema* for a given problem, with a given value of  $q$ , to be an assignment of a set of reducers to each input, subject to the constraints that:

1. No more than  $q$  inputs are assigned to any one reducer.

2. For every output, its associated inputs are all assigned to one reducer. We say the reducer *covers* the output. This reducer need not be unique, and it is, of course, permitted that these same inputs are assigned also to other reducers.

The figure of merit for a mapping schema is the *replication rate*, which we define to be the average number of reducers to which an input is mapped by that schema. Suppose that for a certain algorithm, the  $i$ th reducer is assigned  $q_i \leq q$  inputs, and let  $I$  be the number of different inputs. Then the replication rate  $r$  for this algorithm is

$$r = \sum_{i=1}^p q_i / I$$

We want to derive lower bounds on  $r$ , as a function of  $q$ , for various problems, thus demonstrating the tradeoff between high parallelism (many small reducers) and overhead (total communication cost – the replication rate). These lower bounds depend on counting the total number of outputs that a reducer can cover if it is given at most  $q$  inputs. We let  $g(q)$  denote this number of outputs that a reducer with  $q$  inputs can cover.

Observe that, no matter what random set of inputs is present for an instance of the problem, the expected communication is  $r$  times the number of inputs actually present, so  $r$  is a good measure of the communication cost incurred during an instance of the problem. Further, the assumption that the mapping schema assigns inputs to processors without reference to what inputs are actually present captures the nature of a map-reduce computation. Normally, a map function turns input objects into key-value pairs independently, without knowing what else is in the input.

## 3. THE HAMMING-DISTANCE-1 PROBLEM

We are going to begin our development of the model with the tightest result we can offer. For the problem of finding pairs of bit strings of length  $b$  that are at Hamming distance 1, we have a lower bound on the replication rate  $r$  as a function of  $q$ , the maximum number of inputs assigned to a reducer. This bound is essentially best possible, as we shall point to a number of mapping schemas that solve the problem and have exactly the replication rate stated in the lower bound.

### 3.1 Bounding the Number of Outputs

The key to the lower bound on replication rate as a function of  $q$  is a tight upper bound on the number of outputs that can be covered by a reducer assigned  $q$  inputs.

**LEMMA 3.1.** *For the Hamming-distance-1 problem, a reducer that is assigned  $q$  inputs can cover no more than  $(q/2) \log_2 q$  outputs.*

**PROOF.** The proof is an induction on  $b$ , the length of the bit strings in the input. The basis is  $b = 1$ . Here, there are only two strings, so  $q$  is either 1 or 2. If  $q = 1$ , the reducer can cover no outputs. But  $(q/2) \log_2 q$  is 0 when  $q = 1$ , so the lemma holds in this case. If  $q = 2$ , the reducer can cover at most one output. But  $(q/2) \log_2 q$  is 1 when  $q = 2$ , so again the lemma holds.

Now let us assume the bound for  $b$  and consider the case where the inputs consist of strings of length  $b + 1$ . Let  $X$  be a set of  $q$  bit strings of length  $b + 1$ . Let  $Y$  be the subset of  $X$  consisting of those strings that begin with 0, and let  $Z$  be the remaining strings of  $X$  – those that begin with 1. Suppose  $Y$  and  $Z$  have  $y$  and  $z$  members, respectively, so  $q = y + z$ .

An important observation is that for any string in  $Y$ , there is at most one string in  $Z$  at Hamming distance 1. That is, if  $0w$  is in

$Y$ , it could be Hamming distance 1 from  $1w$  in  $Z$ , if that string is indeed in  $Z$ , but there is no other string in  $Z$  that could be at Hamming distance 1 from  $0w$ , since all strings in  $Z$  start with 1. Likewise, each string in  $Z$  can be distance 1 from at most one string in  $Y$ . Thus, the number of outputs with one string in  $Y$  and the other in  $Z$  is at most  $\min(y, z)$ .

So let's count the maximum number of outputs that can have their inputs within  $X$ . By the inductive hypothesis, there are at most  $(y/2) \log_2 y$  outputs both of whose inputs are in  $Y$ , at most  $(z/2) \log_2 z$  outputs both of whose inputs are in  $Z$ , and, by the observation in the paragraph above, at most  $\min(y, z)$  outputs with one input in each of  $Y$  and  $Z$ .

Assume without loss of generality that  $y \leq z$ . Then the maximum number of strings of length  $b + 1$  that can be covered by a reducer with  $q$  inputs is

$$\frac{y}{2} \log_2 y + \frac{z}{2} \log_2 z + y$$

We must show that this function is at most  $(q/2) \log_2 q$ , or, since  $q = y + z$ , we need to show

$$\frac{y}{2} \log_2 y + \frac{z}{2} \log_2 z + y \leq \frac{y+z}{2} \log_2 (y+z) \quad (1)$$

under the condition that  $z \geq y$ .

First, observe that when  $y = z$ , Equation 1 holds with equality. That is, both sides become  $y(\log_2 y + 1)$ . Next, consider the derivatives, with respect to  $z$ , of the two sides of Equation 1.  $d/dz$  of the left side is

$$\frac{1}{2} \log_2 z + \frac{\log_2 e}{2}$$

while the derivative of the right side is

$$\frac{1}{2} \log_2 (y+z) + \frac{\log_2 e}{2}$$

Since  $z \geq y \geq 0$ , the derivative of the left side is always less than or equal to the derivative of the right side. Thus, as  $z$  grows larger than  $y$ , the left side remains no greater than the right. That proves the induction step, and we may conclude the lemma.  $\square$

### 3.2 The Tradeoff for Hamming Distance 1

We can use Lemma 3.1 to get a lower bound on the replication rate as a function of  $q$ , the maximum number of inputs at a reducer.

**THEOREM 3.2.** *For the Hamming distance 1 problem with inputs of length  $b$ , the replication rate  $r$  is at least  $b/\log_2 q$ .*

**PROOF.** Suppose there are  $p$  reducers, each with  $\leq q$  inputs. By Lemma 3.1, there are at most  $(q_i/2) \log_2 q_i$  outputs covered by reducer  $i$ .

The total number of outputs, given that inputs are of length  $b$  is  $(b/2)2^b$ . Thus, since every output must be covered, and  $\log_2 q \geq \log_2 q_i$  for all  $i$ , we have

$$\sum_{i=1}^p \frac{q_i}{2} \log_2 q_i \geq \frac{b}{2} 2^b \quad (2)$$

$$\sum_{i=1}^p \frac{q_i}{2} \log_2 q \geq \frac{b}{2} 2^b \quad (3)$$

The replication rate is  $r = \sum_{i=1}^p q_i / 2^b$ , that is, the sum of the inputs at each reducer divided by the total number of inputs. We can move factors in Equation 3 to get a lower bound on  $r = \sum_{i=1}^p q_i / 2^b \geq b / \log_2 q$ , which is exactly the statement of the theorem.  $\square$

### 3.3 Upper Bound for Hamming Distance 1

There are a number of algorithms for finding pairs at Hamming distance 1 that match the lower bound of Theorem 3.2. First, suppose  $q = 2$ ; that is, every reducer gets exactly 2 inputs, and is therefore responsible for exactly one output. Theorem 3.2 says the replication rate  $r$  must be at least  $b/\log_2 2 = b$ . But in this case, every input string  $w$  of length  $b$  must be sent to exactly  $b$  reducers – the reducers corresponding to that input and the  $b$  inputs that are Hamming distance 1 from  $w$ .

There is another simple case at the other extreme. If  $q = 2^b$ , then we need only one reducer, which gets all the inputs. In that case,  $r = 1$ . But Theorem 3.2 says that  $r$  must be at least  $b/\log_2(2^b) = 1$ .

In [2], there is an algorithm called Splitting that, for the case of Hamming distance 1 uses  $2^{1+b/2}$  reducers, for some even  $b$ . Half of these reducers, or  $2^{b/2}$  reducers correspond to the  $2^{b/2}$  possible bit strings that may be the first half of an input string. Call these *Group I reducers*. The second half of the reducers correspond to the  $2^{b/2}$  bit strings that may be the second half of an input. Call these *Group II reducers*. Thus, each bit string of length  $b/2$  corresponds to two different reducers.

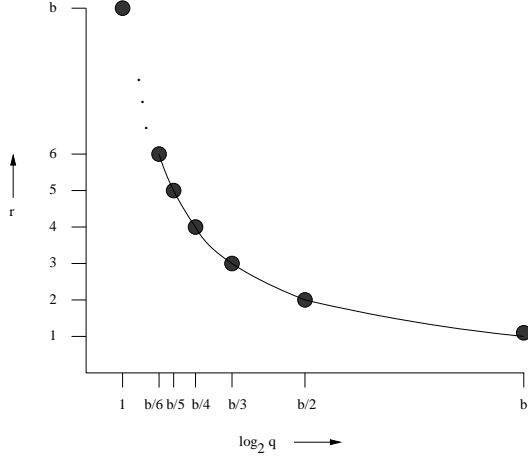
An input  $w$  of length  $b$  is sent to 2 reducers: the Group-I reducer that corresponds to its first  $b/2$  bits, and the Group-II reducer that corresponds to its last  $b/2$  bits. Thus, each input is assigned to two reducers, and the replication rate is 2. That also matches the lower bound of  $b/\log_2(2^{b/2}) = b/(b/2) = 2$ . It is easy to observe that every pair of inputs at distance 1 is sent to some reducer in common. These inputs must either agree in the first half of their bits, in which case they are sent to the same Group-I reducer, or they agree on the last half of their bits, in which case they are sent to the same Group-II reducer.

We can generalize the Splitting Algorithm to give us an algorithm whose replication rate  $r$  matches the lower bound, for any integer  $r > 2$ . We must assume that  $r$  divides  $b$  evenly. Thus, strings of length  $b$  can be split into  $r$  pieces, each of length  $b/r$ . We will have  $r$  groups of reducers, numbered 1 through  $r$ . In each group of reducers there is a reducer corresponding to each of the  $2^{b-b/r}$  bit strings of length  $b - b/r$ .

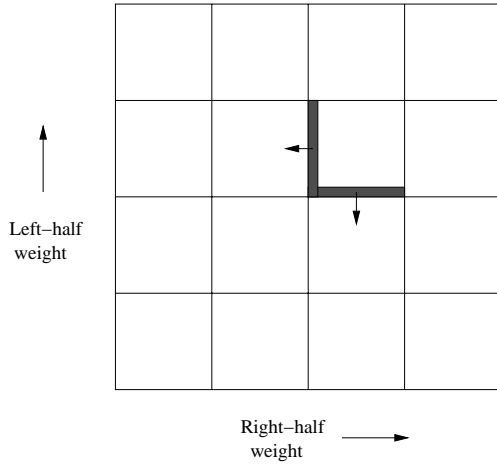
To see how inputs are assigned to reducers, suppose  $w$  is a bit string of length  $b$ . Write  $w = w_1 w_2 \dots w_r$ , where each  $w_i$  is of length  $b/r$ . We send  $w$  to the group- $i$  reducer that corresponds to bit string  $w_1 \dots w_{i-1} w_{i+1} \dots w_r$ , that is,  $w$  with the  $i$ th substring  $w_i$  removed. Thus, each input is sent to  $r$  reducers, one in each of the  $r$  groups, and the replication rate is  $r$ . The input size for each reducer is  $q = 2^{b/r}$ , so the lower bound says that the replication rate must be at least  $b/\log_2(2^{b/r}) = b/(b/r) = r$ . That is the replication rate of our generalization of the Splitting algorithm is tight.

Finally, we need to argue that the mapping schema solves the problem. Any two strings at Hamming distance 1 will disagree in only one of the  $r$  segments of length  $b/r$ . If they disagree in the  $i$ th segments, then they will be sent to the same Group  $i$  reducer, because reducers in this group ignore the value in the  $i$ th segment. Thus, this reducer will cover the output consisting of this pair.

Figure 1 illustrates what we know. The hyperbola is the lower bound. Known algorithms that match the lower bound on replication rate are shown with dots.



**Figure 1: Known algorithms matching the lower bound on replication rate**



**Figure 2: Partitioning by weight. Only the border weights need to be replicated**

### 3.4 An Algorithm for Large $q$

There is a family of algorithms that use reducers with large input  $q$  well above  $2^{b/2}$ , but lower than  $2^b$ . The simplest version of these algorithms divides bit strings of length  $b$  into left and right halves of length  $b/2$  and organizes them by weights, as suggested by Fig. 2. The *weight* of a bit string is the number of 1's in that string. In detail, for some  $k$ , which we assume divides  $b/2$ , we partition the weights into  $b/(2k)$  groups, each with  $k$  consecutive weights. Thus, the first group is weights 0 through  $k-1$ , the second is weights  $k$  through  $2k-1$ , and so on. The last group has an extra weight,  $b/2$ , and consists of weights  $\frac{b}{2} - k$  through  $b/2$ .

There are  $(\frac{b}{2k})^2$  reducers; each corresponds to a range of weights for the first half and a range of weights for the second half. A string is assigned to reducer  $(i, j)$ , for  $i, j = 1, 2, \dots, b/2k$  if the left half of the string has weight in the range  $(i-1)k$  through  $ik-1$  and the right half of the string has weight in the range  $(j-1)k$  through  $jk-1$ .

Consider two bit strings  $w_0$  and  $w_1$  of length  $b$  that differ in exactly one bit. Suppose the bit in which they differ is in the left

half, and suppose that  $w_1$  has a 1 in that bit. Finally, let  $w_1$  be assigned to reducer  $R$ . Then unless the weight of the left half of  $w_1$  is the lowest weight for the left half that is assigned to reducer  $R$ ,  $w_0$  will also be at  $R$ , and therefore  $R$  will cover the pair  $\{w_0, w_1\}$ . However, if the weight of  $w_1$  in its left half is the lowest possible left-half weight for  $R$ , then  $w_0$  will be assigned to the reducer with the same range for the right half, but the next lower range for the left half. Therefore, to make sure that  $w_0$  and  $w_1$  share a reducer, we need to replicate  $w_1$  at the neighboring reducer that handles  $w_0$ . The same problem occurs if  $w_0$  and  $w_1$  differ in the right half, so any string whose right half has the lowest possible weight in its range also has to be replicated at a neighboring reducer. We suggested in Fig. 2 how the strings with weights at the two lower borders of the ranges for a reducer need to be replicated at a neighboring reducer.

Now, let us analyze the situation, including the maximum number  $q$  of inputs assigned to a reducer, and the replication rate. For the bound on  $q$ , note that the vast majority of the bit strings of length  $n$  have weight close to  $n/2$ . The number of bit strings of weight exactly  $n/2$  is  $\binom{n}{n/2}$ . Stirling's approximation [4] gives us  $2^n / \sqrt{2\pi n}$  for this quantity. That is, one in  $O(\sqrt{n})$  of the strings have the average weight.

If we partition strings as suggested by Fig. 2, then the most populous  $k \times k$  cell, the one that contains strings with weight  $b/4$  in the first half and also weight  $b/4$  in the second half, will have no more than

$$k^2 \left( \frac{2^{b/2}}{\sqrt{2\pi(b/2)}} \right)^2 = \frac{k^2 2^b}{\pi b}$$

strings assigned.<sup>1</sup> If  $k$  is a constant, then in terms of the horizontal axis in Fig. 1, this algorithm has  $\log_2 q$  equal to  $b - \log_2 b$  plus or minus a constant. It is thus very close to the right end, but not exactly at the right end.

For the replication rate of the algorithm, if  $k$  is a constant, then for any cell there is only a small ratio of variation between the numbers of strings with weights  $i$  and  $j$  in the left and right halves, for any  $i$  and  $j$  that are assigned to that cell. Moreover, when we look at the total number of strings in the borders of all the cells, the differences average out so the total number of replicated strings is very close to  $(2k)/k^2 = 2/k$ . That is, a string is replicated if either its left half has a weight divisible by  $k$  or its right half does. Note that strings in the lower-left corner of a cell are replicated twice, strings of the other  $2k-2$  points on the border are replicated once, and the majority of strings are not replicated at all. We conclude that the replication rate is  $1 + \frac{2}{k}$ .

### 3.5 Generalization to $d$ Dimensions

The algorithm of Section 3.4 can be generalized from 2 dimensions to  $d$ . Break bit strings of length  $b$  into  $d$  pieces of length  $b/d$ , where we assume  $d$  divides  $b$ . Each string of length  $b$  can thus be assigned to a cell in a  $d$ -dimensional hypercube, based on the weights of each of its  $d$  pieces. Assume that each cell has side  $k$  in each dimension, where  $k$  is a constant that divides  $b/d$ .

The most populous cell will be the one that contains strings where each of its  $d$  pieces has weight  $b/(2d)$ . Again using Stir-

<sup>1</sup>Note that many of the cells have many fewer strings assigned, and in fact a fraction close to 1 of the strings have weights within  $\sqrt{b}$  of  $b/4$  in both their left halves and right halves. In a realistic implementation, we would probably want to combine the cells with relatively small population at a single reducer, in order to equalize the work at each reducer.

ling's approximation, the number of strings assigned to this cell is

$$k^d \left( \frac{2^{b/d}}{\sqrt{2\pi b/d}} \right)^d = \frac{k^d 2^b}{b^{d/2} (2\pi/d)^{d/2}}$$

On the assumption that  $k$  is constant, the value of  $\log_2 q$  is

$$b - (d/2) \log_2 b$$

plus or minus a constant.

To compute the replication rate, observe that every point on each of the  $d$  faces of the hypercube that are at the low ends of their dimension must be replicated. The number of points on one face is  $k^{d-1}$ , so the sum of the volumes of the faces is  $dk^{d-1}$ . The entire volume of a cell is  $k^d$ , so the fraction of points that are replicated is  $d/k$ , and the replication rate is  $1 + d/k$ . Technically, we must prove that the points on the border of a cell have, on average, the same number of strings as other points in the cell. As in Section 3.4, the border points in any dimension are those whose corresponding substring has a weight divisible by  $k$ . As long as  $k$  is much smaller than  $b/d$ , this number is close to  $1/k$ th of all the strings of that length.

## 4. TRIANGLE FINDING

In this section, we present a brief description of other results obtained using our framework, specifically on finding triangles. The pattern that lets us investigate any problem is, we hope, clear from the analysis of Section 3.

1. Find an upper bound,  $g(q)$ , on the number of outputs a reducer can cover if  $q$  is the number of inputs it is given.
2. Count the total numbers of inputs  $|I|$  and outputs  $|O|$ .
3. Assume there are  $p$  reducers, each receiving  $q_i \leq q$  inputs and covering  $g(q_i)$  outputs. Together they cover all the outputs. That is  $\sum_{i=1}^p g(q_i) \geq |O|$ .
4. Manipulate the inequality from (3) to get a lower bound on the replication rate, which is  $\sum_{i=1}^p q_i / |I|$ .
5. Hopefully, demonstrate that there are algorithms whose replication rate matches the formula from (4).

### 4.1 The Tradeoff

We shall briefly show how this method applies to the problem of finding triangles introduced in Example 2.2. Suppose  $n$  is the number of nodes of the input graph. Following the outline just given:

1. We claim that the largest number of outputs (triangles) a reducer with at most  $q$  inputs occurs when the reducer is assigned all the edges running between some set of  $k$  nodes. This point was proved, to within an order of magnitude in [5]. Suppose we assign to a reducer all the edges between a set of  $k$  nodes. Then there are  $\binom{k}{2}$  edges assigned to this reducer, or approximately  $k^2/2$  edges. Since this quantity is  $q$ , we have  $k = \sqrt{2q}$ . The number of triangles among  $k$  nodes is  $\binom{k}{3}$ , or approximately  $k^3/6$  outputs. In terms of  $q$ , the upper bound on the number of outputs is  $\frac{\sqrt{2}}{3} q^{3/2}$ .
2. The number of inputs is  $\binom{n}{2}$  or approximately  $n^2/2$ . The number of outputs is  $\binom{n}{3}$ , or approximately  $n^3/6$ .
3. So using the formulas from (1) and (2), if there are  $p$  reducers each with  $\leq q$  inputs:  $\sum_{i=1}^p \frac{\sqrt{2}}{3} q_i^{3/2} \geq n^3/6$ , which implies that  $\sum_{i=1}^p \frac{\sqrt{2}}{3} q_i^{1/2} \geq n^{3/6}$ .

4. The replication rate is  $\sum_{i=1}^p q_i$  divided by the number of inputs, which is  $n^2/2$  from (1). We can manipulate the inequality from (3) to get

$$r = \frac{2 \sum_{i=1}^p q_i}{n^2} \geq \frac{n}{\sqrt{2q}}$$

5. There are known algorithms that, to within a constant factor, match the lower bound on replication rate. See [6] and [1].<sup>2</sup>

**Generalizing to multiway joins** Finding triangles is equivalent to computing the multiway join  $E(A, B) \Join E(B, C) \Join E(C, A)$ . Similar techniques can be used to compute lower and upper bounds for any multiway join. In particular, in the case where we have one relation of arity  $a$  and the multiway join uses  $m$  variables then we get lower and upper bounds that are both  $O(q^{1-m/a} n^{m-a})$ .

## 5. SUMMARY

This abstract introduced a simple model for defining map-reduce problems, enabling us to study their “distributability” properties. We studied the notion of *replication rate*, which is closely related communication cost, and the number of machines available for mappers and reducers. We showed that our model effectively captures a multitude of map-reduce problems, and is a natural formalism for the study of replication rate. We presented a detailed treatment of the hamming-distance-1 problem, providing tight bounds on the replication rate. We also presented a summary of some other results on multiway joins and triangle finding we have obtained.

We believe that our formalism presents a new direction for the study of a large class of map-reduce problems, and allows us to prove results on the limits of map-reducibility for any algorithm for a problem that fits our model.

## 6. REFERENCES

- [1] Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. Enumerating subgraph instances using map-reduce. Technical report, January 2012. Available at <http://ilpubs.stanford.edu:8090/1020/>.
- [2] Foto N. Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, and Jeffrey D. Ullman. Fuzzy joins using mapreduce. In *ICDE '12*.
- [3] Robson Leonardo Ferreira Cordeiro, Caetano Traina Jr., Agma Juci Machado Traina, Julio López, U. Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *KDD*, 2011.
- [4] W. Feller. *An Introduction to Probability Theory and Its Applications*. Vol. 1, 3rd ed. New York: Wiley, 1968. (Stirling's Formula in Section 2.9).
- [5] T. Schank. *Algorithmic Aspects of Triangle-Based Network*. University of Karlsruhe (TH), 2007.
- [6] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW '11*.
- [7] W-C. Tan. Provenance in Databases: Past, Current, and Future. *IEEE Data Engineering Bulletin*, 2008.
- [8] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD Conference*, 2010.

<sup>2</sup>It is a little tricky to relate these algorithms to the bound, since those algorithms assume the actual data graphs are sparse and calculate replication and input sizes in terms of the number of edges rather than nodes. However, on randomly chosen subsets of all possible edges, they do get us within a constant factor of the lower bound.